Deploying Custom Object Detection Model on `OAK-D-Lite` - TensorFlow and Keras

Hardware Setup:

Specifications of `OAK-D-Lite` by Luxonis: https://docs-old.luxonis.com/projects/hardware/en/latest/pages/DM9095/

- We will use a OAK-D-Lite with `AutoFocus` (`13MP IMX214 Color Camera`, `MaxFrameRate - 35 FPS`)
- Power Consumption: **USB 3 C-Type** Cable with `900mA at 5V`
- OAK-D-Lite has onboard `Intel® Movidius™ *Myriad*™ X` vision processor (`VPU`) which is capable of doing `4 Trillion` neural operations per sec.

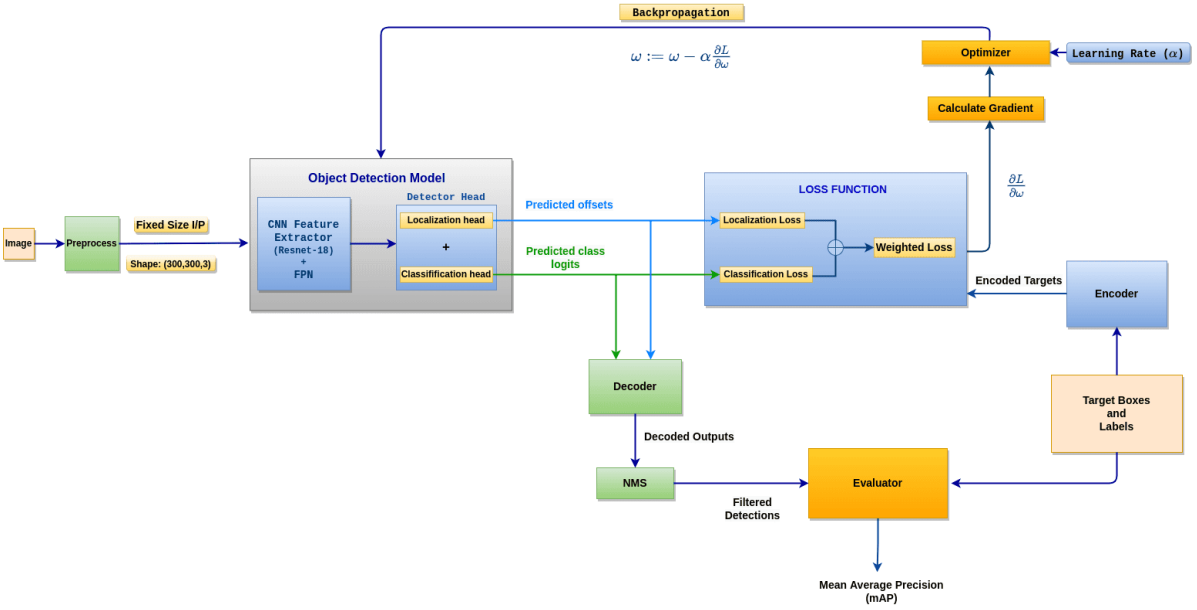
Custom Object Detection Module - TensorFlow Course

Here we will deploy the Custom `Object Detection Model` that we have trained on `PenFudan Dataset`.

Module 06 - Object Detection- Create Custom Object Detector - Training from Scratch

- For this experiment we will use a custom detector with `Resnet-18` backbone with two heads, one for `classification` and other for `localization`.

  Here's an **overview** of our training pipeline:



Some `Scripts` that have been modified from Course Module to adapt to `OpenVINO 2022.1` Toolkit:

fpn.py

For `OpenVINO 2022.1` to work as intended it requires an `opset` version more than `8`, so we have to ensure all our model layers are supported in OpenVINO layers.

Possible Error: `ERROR - Tensorflow op [StatefulPartitionedCall/custom_model/fpn/lateral__connection/resize/ScaleAndTranslate: ScaleAndTranslate] is not supported`

To fix this, we will make a slight modification in `tf.image.resize` with a `bilinear` interpolation (Resize method) instead of `lanczos5` and set the default `antialias=False`.

```
class Lateral_Connection(tf.keras.layers.Layer):
    def __init__(self, channels_out, **kwargs):
      super().__init__(**kwargs)
      self.conv_1x1 = Conv2D(channels_out, 1, 1, "same")
    def call(self, inputs):
      prev, current = inputs
      up = tf.image.resize(prev, size=tf.shape(current)[1:3], method='bilinear')  # Modified line
      current = self.conv_1x1(current)
      x = up + current
      return x
```

Detector.py

The scripts of `detector.py` was modified to address the following error,

`ValueError: '_class/' is not a valid root scope name. A root scope name has to match the following pattern: ^[A-Za-z0-9.][A-Za-z0-9_.\\/>-]*$`

To fix this,

```
 class DetectorHead(Layer):
    def __init__(self, fpn_channels=64, num_anchors=9, num_classes=2, localization=True, **kwargs):
       ...
        if self.localization:
            self.output_head_nodes = 4  # Co-ordinates (x1, y1, x2, y2)
            self.head = DetectorHead._make_head(self.fpn_channels, self.num_anchors *
self.output_head_nodes) # Modified line
            # ** name="_local" attrbute is removed **
        else:
            self.output_head_nodes = num_classes
            self.head = self._make_head(self.fpn_channels, self.num_anchors * self.output_head_nodes)
# Modified line
            #  ** name="_class" attribute is removed *
```

In the Notebook for our deployment task, instead of saving the model as `.ckpt`, we will save directly to tensorflow saved model (`.pb`) format. This is done to easily convert between supported model formats like `.onxx`, `.keras`, or `OpenVINO IR`.

We change the following in the Training from scratch notebook,

9. **Defining Callbacks Section**

```
class SaveBestModel(tf.keras.callbacks.Callback):
    def __init__(self, save_path, monitor='val_mAP', mode='max', verbose=1):
        super(SaveBestModel, self).__init__()
        self.save_path = save_path
        self.monitor = monitor
        self.mode = mode
        self.verbose = verbose
        self.best = -np.inf if mode == 'max' else np.inf

    def on_epoch_end(self, epoch, logs=None):
        current_value = logs.get(self.monitor)
        if self.mode == 'max' and current_value > self.best:
            self.best = current_value
            self.model.save(self.save_path, save_format='tf', overwrite=True)
            if self.verbose > 0:
                print(f"\nEpoch {epoch+1}: {self.monitor} improved to {current_value:.4f}, saving
model to {self.save_path}.")
        elif self.mode == 'min' and current_value < self.best:
            self.best = current_value
            self.model.save(self.save_path, save_format='tf', overwrite=True)
            if self.verbose > 0:
                print(f"\nEpoch {epoch+1}: {self.monitor} improved to {current_value:.4f}, saving
model to {self.save_path}.")

def get_callbacks(training_config):
    log_dir = training_config.log_dir
    tensorboard_callback = tf.keras.callbacks.TensorBoard(
        log_dir=log_dir,
        histogram_freq=20,
        write_graph=False,
        update_freq="epoch",
        write_images=True,
    )

    # Updated save path for the full model
    save_path = os.path.join(training_config.checkpoint_dir, "saved_model")

    # Custom callback to save the best model
    save_best_model_callback = SaveBestModel(
        save_path=save_path,
        monitor='val_mAP',
        mode='max',
```

```
        verbose=1,
    )

    return [tensorboard_callback, save_best_model_callback
```

- The code is slightly modified to save model based on best `mAP` in `tf saved model` format.
- We recommend using `Tensorflow==2.15.1` version and `keras==2.15.0` version to avoid code break.

That's it! Now we are ready to train again for `150 epochs` with this slightly modified code blocks.

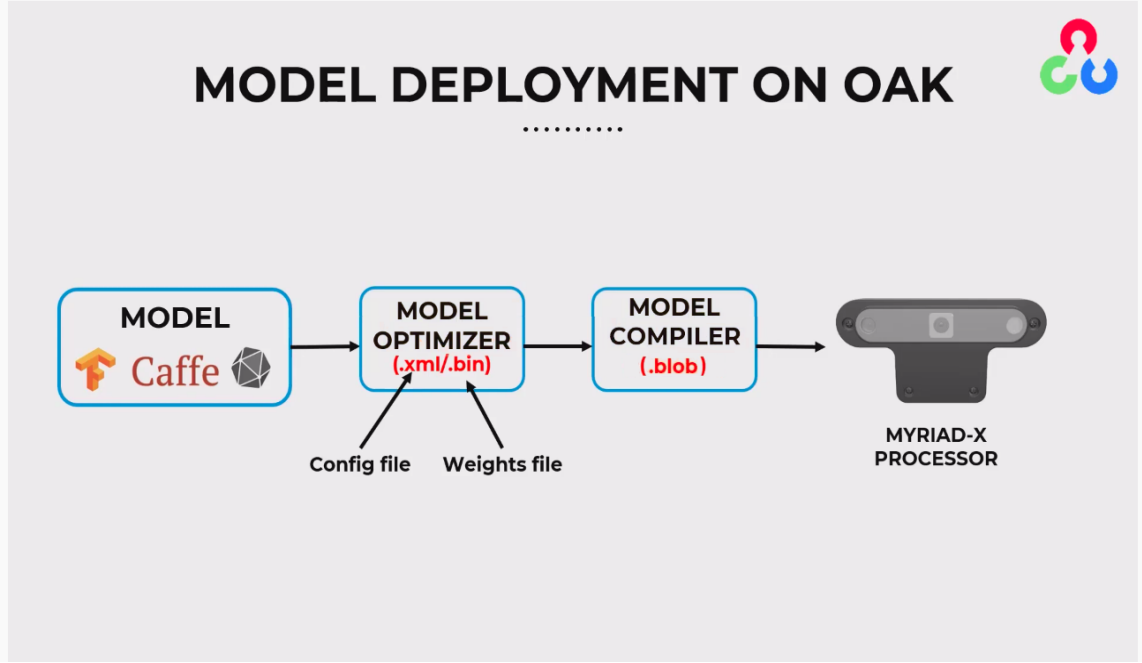
Model Conversion

To use our custom model on OAK_D, it has to be in `.blob` model format for an optimized DepthAI inference.

Also for custom model deployment on OAK-D, the documentation suggests to first convert the `tf saved .pb model` to `.onxx` format then to `OpenVINO IR` (Intermediate Representation) and then to `.blob` format.

Why OpenVINO?

Under-the-hood, `DepthAI` (OAK-D) uses the Intel technology to perform high-speed model inference. However, you can't just dump your neural net into the chip and get high-performance for free. That's where OpenVINO comes in. OpenVINO is a free toolkit that converts a deep learning model into a format that runs on Intel Hardware. Once the model is converted, it's common to see `Frames Per Second (FPS)` improve by 25x or more. Are a couple of small steps worth a 25x FPS increase? Often, the answer is yes!



Onxx Conversion

In a Jupyter or Colab notebook,

```
!pip install -q tf2onnx onnxruntime
```

```
!python -m tf2onnx.convert --saved-model Logs_Checkpoints/Model_checkpoints/version_5/saved_model --
output resnet18_final_150epochs_saved_model.onnx --opset 11
```

The `.onxx` model is saved.

We know that we have the batch size to be dynamic while defining the shape for our tensorflow model training.

Here we can see the batch size with a placeholder like `unk__595` for dynamic batch input.

```
Model inputs:
NodeArg(name='input_3', type='tensor(float)', shape=['unk__595', 300, 300, 3])
Model outputs:
NodeArg(name='cls_head', type='tensor(float)', shape=['unk__596', 7756, 2])
NodeArg(name='loc_head', type='tensor(float)', shape=['unk__597', 7756, 4])
```

To avoid `dynamic shape` error from OpenVINO we will make model to have a **fixed input dimension**.

It is recommended to do a sanity check by visualizing the predictions and input or output shape of the converted model.

To convert the **OpenVINO IR** format we need to install the **openvino-dev toolkit**,

```
!pip install -q openvino-dev==2022.3.0
```

Then using **model optimizer** (mo) from openvino-dev we will convert to IR format.

```
!mo --input_model resnet18_final_150epochs_saved_model.onnx --input_shape [1,300,300,3] --data_type
FP16 --source_layout nhwc --target_layout nchw
```

Some key things have to be handled carefully here:

- The model is now converted to have a fixed input dimension.
- Our `depthai` by default expect the model input frame dimensions to follow `NCHW` i.e., (no. of Batches, no. of channels, Height, Width) like `Pytorch`.
    - Contrarily in `Tensorflow` model's dimensions are in `NHWC`, so lets convert that in the cli argument using `--source_layout` (TF Input Format) and `--target_layout` (DepthAI Input Format).

After executing this command , a `.xml` (configuration file) and `.bin` (weights file) are saved, which is the desired OpenVINO IR Format.

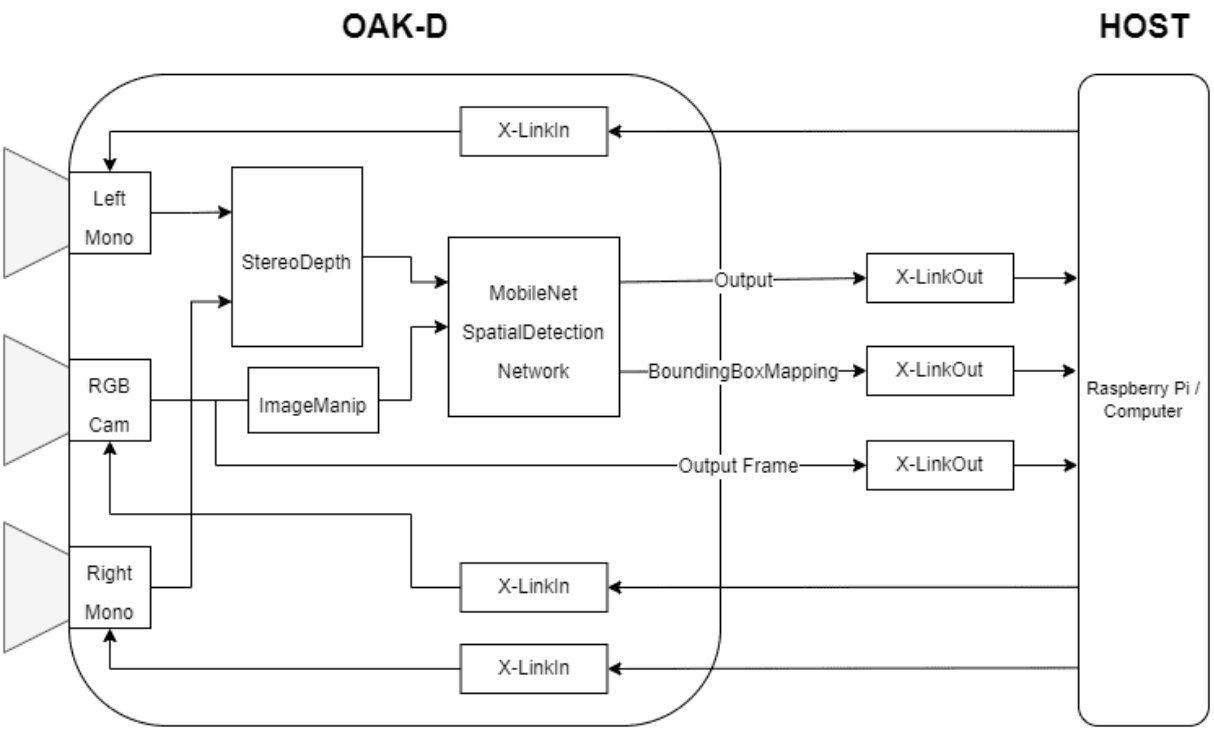
Luxonis Blob Converter

Final steps in `depthai` compatible model preparation is the `.blob` format conversion.

- Visit the Luxonis official website for `.blob` converter : https://blobconverter.luxonis.com/
- Choose the OpenVINO version to be `2022.1` and Model source to be `OpenVINO` and hit the continue button.
- Now in the next page, upload your `.xml` and `.bin` files of your model and use the SHAVES to `13`
- **SHAVES** refer to the computational units within the Myriad X VPU (Vision Processing Unit) that execute neural network inferences, with more shaves allowing for faster processing.
- We will leave the default compile parameters and the `.blob` is downloaded.

Setting up OAK-D-Lite Pipeline

The image shown here is a pipeline for a pretrained Mobilenet Model from Zoo - Illustration Image:



Courtesy: https://learnopencv.com/object-detection-with-depth-measurement-with-oak-d/

We will slightly modify this pipeline for our custom Object detection model.

At first:

```
!python3 -m pip install depthai
```

Test your depthai installation:

```
https://docs.luxonis.com/software/depthai/manual-install/#Test%20Installation
```

Import Dependencies

```
from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np
import time
from encoder import DataEncoder

H, W = 300, 300

nnPath = "oak_d/NMS/U8_resnet18_final_150epochs_saved_model.blob" #Adjust your model patha accordingly

if not Path(nnPath).exists():
    raise FileNotFoundError(f'Required file/s not found, please check the path: {nnPath}')

# Resnet-18 Pedestrian label texts
labelMap = ["__background__", "Person"]
```

Visualize Predictions

```
def visualize_predictions(frame, loc_pred, cls_pred, encoder, labelMap):
    color = (255, 0, 0)
    imH, imW = frame.shape[:2]
    IMG_SIZE_H, IMG_SIZE_W = encoder.input_size

    ratio_h = imH / IMG_SIZE_H
    ratio_w = imW / IMG_SIZE_W

    loc_pred = np.array(loc_pred).reshape(-1, 4)
    cls_pred = np.array(cls_pred).reshape(-1, len(labelMap))

    preds = encoder.decode(loc_pred, cls_pred,nms_threshold=0.3,score_threshold=0.7,soft_nms_sigma =
0.3 ).numpy()

    for class_idx, class_name in enumerate(labelMap):
        if class_name == "__background__":
            continue

        class_tensor = preds[np.where(preds[:, 5] == class_idx)]

        class_tensor[:, 0] = np.maximum(0, (class_tensor[:, 0] * ratio_w))
        class_tensor[:, 1] = np.maximum(0, (class_tensor[:, 1] * ratio_h))
        class_tensor[:, 2] = np.minimum(imW, (class_tensor[:, 2] * ratio_w))
        class_tensor[:, 3] = np.minimum(imH, (class_tensor[:, 3] * ratio_h))

        pred_labels = [f"{labelMap[int(i)]}" for i in class_tensor[:, 5]]

        for box, label in zip(class_tensor, pred_labels):
            x_min, y_min, x_max, y_max = box[:4].astype(int)
            cv2.rectangle(frame, (x_min, y_min), (x_max, y_max), color, 2)
            cv2.putText(frame, label, (x_min, y_min - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
            cv2.putText(frame, f"{box[4] * 100:.2f}%", (x_min, y_min - 30), cv2.FONT_HERSHEY_SIMPLEX,
0.5, color, 2)

    return frame
```

- The function takes in live camera frames, detected object bounding box locations, and class predictions for all those detected objects.
- We will have to resize our `bounding box` according to model encoder input size. i.e., `300,300`.

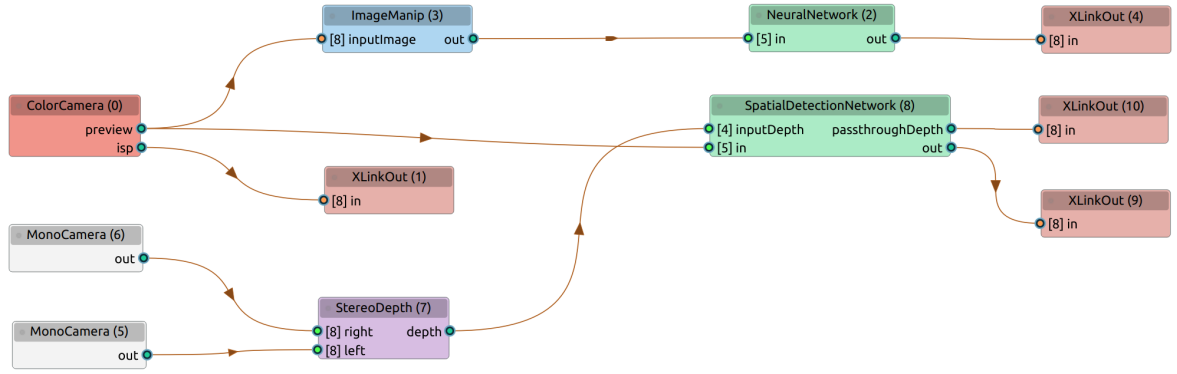
```
class FPSHandler:
    def __init__(self, cap=None):
        self.timestamp = time.time()
        self.start = time.time()
        self.frame_cnt = 0

    def next_iter(self):
        self.timestamp = time.time()
        self.frame_cnt += 1

    def fps(self):
        return self.frame_cnt / (self.timestamp - self.start)
```

We will initialise our encoder with `input size` and `labelMap`.

```
# Initialize DataEncoder with input size and labels
encoder = DataEncoder(input_size=(H, W), classes=labelMap)
```

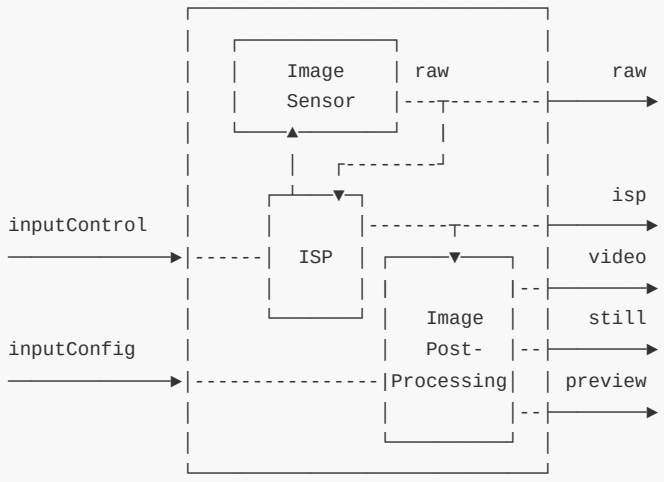
We will map the `class index` with the `labelMap`.

DepthAI Pipeline



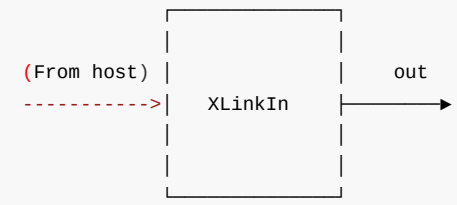
Courtesy: DepthAI Pipeline Github - Illustration shown for Demo purposes only

`NN` means Neural Network (i.e., Our Model)

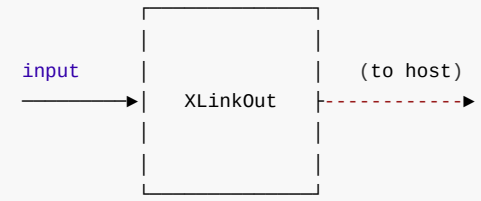



```
syncNN = False

pipeline = dai.Pipeline()
# Create a pipeline object to define the processing pipeline

camera = pipeline.create(dai.node.ColorCamera)
# Create a ColorCamera node for capturing color images
camera.setPreviewSize(H, W)
# Set the preview size of the camera to the specified height (H) and width (W)
camera.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
# Set the camera resolution to 1080p
camera.setInterleaved(False)
# Disable interleaved mode, so images are not interleaved
camera.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)
# Set the color order to BGR (Blue, Green, Red)

nn = pipeline.create(dai.node.NeuralNetwork)
# Create a NeuralNetwork node for running the neural network model
nn.setBlobPath(nnPath)
# Set the path to the neural network model blob file
nn.setNumInferenceThreads(2)
# Set the number of inference threads to 2 for parallel processing
nn.input.setBlocking(True)
```

```
# Set the input to the neural network node to be blocking

# Send camera frames to the host
camera_xout = pipeline.create(dai.node.XLinkOut)
# Create an XLinkOut node to send data from the camera to the host
camera_xout.setStreamName("camera")
# Set the stream name for the camera output
camera.preview.link(camera_xout.input)
# Link the camera preview output to the XLinkOut input

# Send converted frames from the host to the NN
nn_xin = pipeline.create(dai.node.XLinkIn)
/
```

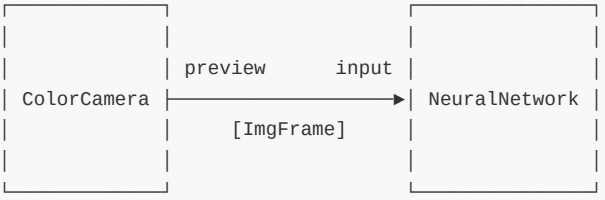



```
# Create an XLinkIn node to receive data from the host and send it to the NN
nn_xin.setStreamName("nnInput")
# Set the stream name for the NN input
nn_xin.out.link(nn.input)
# Link the XLinkIn output to the NN input

# Send bounding boxes from the NN to the host via XLink
nn_xout = pipeline.create(dai.node.XLinkOut)
# Create an XLinkOut node to send NN output (bounding boxes) to the host
nn_xout.setStreamName("nn")
# Set the stream name for the NN output
/
```




```
# Passing NN output to the host
nn.out.link(nn_xout.input)
# Link the NN output to the XLinkOut input
```

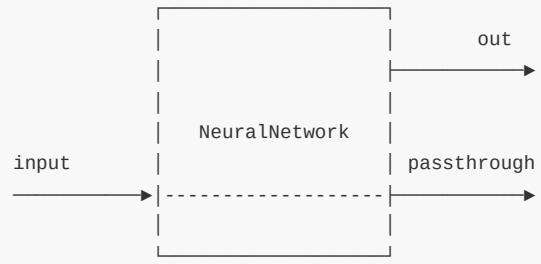
Setting `nn.input.setBlocking(True)` ensures that the input queue to the neural network node blocks if it is full, meaning it waits for space to be available before adding more frames, while `syncNN = False` ensures that the neural network operates asynchronously, processing frames as they arrive rather than synchronizing with the input stream.



```
dai_frame = dai.ImgFrame()
# Create an ImgFrame object to hold image data

dai_frame.setType(dai.ImgFrame.Type.BGRF16F16F16i)
# Set the type of the image frame to BGRF16F16F16i, which indicates a BGR format with 16-bit floating
point precision for each channel

dai_frame.setHeight(H)
# Set the height of the image frame to the specified value H

dai_frame.setWidth(W)
# Set the width of the image frame to the specified value W
```

More about `depthai ImgFrame` : https://docs.luxonis.com/software/depthai-components/messages/img_frame

```
# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Connect to the device using the defined pipeline in a context manager

    qCamera = device.getOutputQueue(name="camera", maxSize=4, blocking=True)
    # Create an output queue named "camera" to receive frames from the camera node
    # maxSize=4 sets the maximum number of frames to be held in the queue
    # blocking=True means the queue will block if it's full until space is available

    qNnInput = device.getInputQueue(name="nnInput", maxSize=20, blocking=True)
    # Create an input queue named "nnInput" to send frames to the neural network node
    # maxSize=20 sets the maximum number of frames to be held in the queue
    # blocking=True means the queue will block if it's full until space is available

    qNn = device.getOutputQueue(name="nn", maxSize=4, blocking=True)
    # Create an output queue named "nn" to receive the output from the neural network node
    # maxSize=4 sets the maximum number of results to be held in the queue
    # blocking=True means the queue will block if it's full until space is available

    fps = FPSHandler()
    # Instantiate an FPSHandler to keep track of the frames per second if required.
```




```
# Enter an infinite loop to continuously process frames
while True:
    # Retrieve a new frame from the camera queue
    inRgb = qCamera.get()

    # Convert the frame to an OpenCV-compatible format
    frame = inRgb.getCvFrame()

    # Print the shape of the current frame for debugging
    print(frame.shape)

    # Change data layout of the image from HWC (Height, Width, Channels) to CHW (Channels, Height,
Width)
    input_data = frame.transpose(2, 0, 1)
```

```
    # Create a new ImgFrame object to send to the neural network
    dai_frame = dai.ImgFrame()

    # Set the data for the ImgFrame object with the transposed image data
    dai_frame.setData(input_data)

    # Specify the type of the ImgFrame as BGR888p
    dai_frame.setType(dai.ImgFrame.Type.BGR888p)

    # Set the width of the ImgFrame
    dai_frame.setWidth(W)

    # Set the height of the ImgFrame
    dai_frame.setHeight(H)

    # Send the ImgFrame to the neural network input queue
    qNnInput.send(dai_frame)

    # Retrieve the output from the neural network queue
    in_nn = qNn.get()
```

```
        # Check if the neural network output is not None
        if in_nn is not None:
            # Update the FPS counter
            fps.next_iter()

            # Display the current FPS on the frame
            cv2.putText(frame, "Fps: {:.2f}".format(fps.fps()), (2, 300 - 4), cv2.FONT_HERSHEY_TRIPLEX,
0.4, color=(255, 255, 255))

            # Get the location predictions from the neural network output
            loc_pred = in_nn.getLayerFp16('loc_head')

            # Get the classification predictions from the neural network output
            cls_pred = in_nn.getLayerFp16('cls_head')

            # Reshape the location predictions to the shape (7756, 4)
            loc_pred_reshaped = np.reshape(np.array(loc_pred, dtype=np.float32), (7756, 4))

            # Reshape the classification predictions to the shape (7756, 2)
            cls_ped_reshaped = np.reshape(np.array(cls_pred, dtype=np.float32), (7756, 2))

            # Print the shape of the reshaped location predictions for debugging
            print("Loc Pred: ", loc_pred_reshaped.shape)

            # Print the shape of the reshaped classification predictions for debugging
            print("Cls Pred: ", cls_ped_reshaped.shape)

            frame = visualize_predictions(frame, loc_pred_reshaped, cls_ped_reshaped, encoder,
labelMap)

        cv2.imshow("Camera", frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

# Release resources
cv2.destroyAllWindows()
```

The original `tf saved model` had best `val mAP` of `88.68`, and after converting it to `.blob` format we noticed a slight drop in `mAP` to be `85.96` which is actually good for a quantized model.

Observations:

- We got on a average of `15` fps in OAK-D-Lite with a USB 3 C Type thunderbolt cable.
- To avoid False Positives we adjust the `NMS threshold` of `0.3` and `confidence threshold` of `0.7`

Congrats on deploying custom model on your OAK-D-Lite Device successfully!

APPENDIX:

Model Conversion

- Converting to Onnx here: https://onnx.ai/
- Saving Tensorflow Model with Custom Layers : Tensorflow Docs

Luxonis BlobConverter

- **Blob Model Conversion** for depthai (OAK-D): https://docs.luxonis.com/software/ai-inference/conversion
- To understand more how shave size is determined based on model and camera resolution, have a look at this discussion:

  https://discuss.luxonis.com/d/1122-how-many-shaves-available-in-oak-d-pipeline

---

Optimization Techniques:

If your looking for optimizing FPS: https://docs.luxonis.com/software/depthai/optimizing/

More on OpenVINO post processing quantization techniques: https://docs.openvino.ai/2023.3/ptq_introduction.html

OpenVINO

- More about Opset-Operations in OpenVINO : https://docs.openvino.ai/2023.3/openvino_docs_MO_DG_IR_and_opsets.html
- How to find the correct opset version: https://docs.openvino.ai/2023.3/openvino_docs_ops_opset.html
- OpenVINO for DepthAI: Luxonis Docs

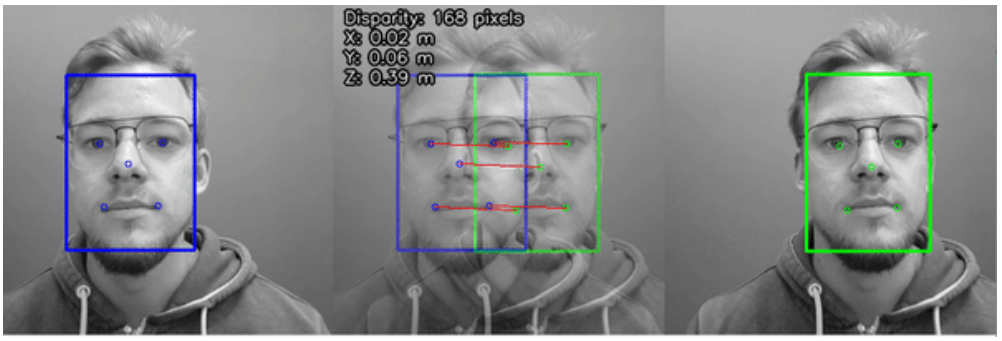
Setting up OAK-D (Pre-trained Models): (Last)

Recommended LearnOpenCV articles to start with: BlogPost

- Intro to OAK-D: https://learnopencv.com/introduction-to-opencv-ai-kit-and-depthai/
- Object Detection with Pretrained Models: https://learnopencv.com/object-detection-with-depth-measurement-with-oak-d/
- Face Detection : https://learnopencv.com/anti-spoofing-face-recognition-system-using-oak-d-and-depthai/
- Custom Pipeline: https://learnopencv.com/depthai-pipeline-overview-creating-a-complex-pipeline/
- Stereo & Depth Estimation: https://learnopencv.com/stereo-vision-and-depth-estimation-using-opencv-ai-kit/

OpenCV AI Competition 2021

- Interesting Applications with OAK-D: https://opencv.org/opencv-ai-competition-2021/



DepthAI - Github

- DepthAI - Community Experiments: https://github.com/luxonis/depthai-experiments?tab=readme-ov-file
- DepthAI Repo: https://github.com/luxonis/depthai